O'REILLY®

Flask Web Development

DEVELOPING WEB APPLICATIONS WITH PYTHON

Miguel Grinberg

Flask Web Development

by Miguel Grinberg

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Meghan Blanchette and Rachel Roumeliotis
Production Editor: Nicole Shelby
Copyeditor: Nancy Kotary
Proofreader: Charles Roumeliotis

Cover Designer: Randy Comer
Interior Designer: David Futato
Illustrator: Rebecca Demarest

May 2014: First Edition

Revision History for the First Edition:

2014-04-25: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449372620* for release details.

ISBN: 978-1-449-37262-0

[LSI]

Table of Contents

Preface xi

Part I. Introduction to Flask

1. Installation 3
Using Virtual Environments 4
Installing Python Packages with pip 6

2. Basic Application Structure 7
Initialization 7
Routes and View Functions 8
Server Startup 9
A Complete Application 9
The Request-Response Cycle 12
Application and Request Contexts 12
Request Dispatching 14
Request Hooks 14
Responses 15
Flask Extensions 16
Command-Line Options with Flask-Script 17

3. Templates 21
The Jinja2 Template Engine 22
Rendering Templates 22
Variables 23
Control Structures 24
Twitter Bootstrap Integration with Flask-Bootstrap 26
Custom Error Pages 29
Links 31

Static Files 32
Localization of Dates and Times with Flask-Moment 33

4. Web Forms. 37
Cross-Site Request Forgery (CSRF) Protection 37
Form Classes 38
HTML Rendering of Forms 40
Form Handling in View Functions 41
Redirects and User Sessions 44
Message Flashing 46

5. Databases. 49
SQL Databases 49
NoSQL Databases 50
SQL or NoSQL? 51
Python Database Frameworks 51
Database Management with Flask-SQLAlchemy 52
Model Definition 54
Relationships 56
Database Operations 57
Creating the Tables 58
Inserting Rows 58
Modifying Rows 60
Deleting Rows 60
Querying Rows 60
Database Use in View Functions 62
Integration with the Python Shell 63
Database Migrations with Flask-Migrate 64
Creating a Migration Repository 64
Creating a Migration Script 65
Upgrading the Database 66

6. Email. 69
Email Support with Flask-Mail 69
Sending Email from the Python Shell 70
Integrating Emails with the Application 71
Sending Asynchronous Email 72

7. Large Application Structure. 75
Project Structure 75
Configuration Options 76
Application Package 78

Using an Application Factory 78
Implementing Application Functionality in a Blueprint 79
Launch Script 81
Requirements File 82
Unit Tests 83
Database Setup 85

Part II. Example: A Social Blogging Application

8. User Authentication. 89
Authentication Extensions for Flask 89
Password Security 90
Hashing Passwords with Werkzeug 90
Creating an Authentication Blueprint 92
User Authentication with Flask-Login 94
Preparing the User Model for Logins 94
Protecting Routes 95
Adding a Login Form 96
Signing Users In 97
Signing Users Out 99
Testing Logins 99
New User Registration 100
Adding a User Registration Form 100
Registering New Users 102
Account Confirmation 103
Generating Confirmation Tokens with itsdangerous 103
Sending Confirmation Emails 105
Account Management 109

9. User Roles. 111
Database Representation of Roles 111
Role Assignment 113
Role Verification 114

10. User Profiles. 119
Profile Information 119
User Profile Page 120
Profile Editor 122
User-Level Profile Editor 122
Administrator-Level Profile Editor 124

User Avatars 127

11. Blog Posts. 131
Blog Post Submission and Display 131
Blog Posts on Profile Pages 134
Paginating Long Blog Post Lists 135
Creating Fake Blog Post Data 135
Rendering Data on Pages 137
Adding a Pagination Widget 138
Rich-Text Posts with Markdown and Flask-PageDown 141
Using Flask-PageDown 141
Handling Rich Text on the Server 143
Permanent Links to Blog Posts 145
Blog Post Editor 146

12. Followers. 149
Database Relationships Revisited 149
Many-to-Many Relationships 150
Self-Referential Relationships 151
Advanced Many-to-Many Relationships 152
Followers on the Profile Page 155
Query Followed Posts Using a Database Join 158
Show Followed Posts on the Home Page 160

13. User Comments. 165
Database Representation of Comments 165
Comment Submission and Display 167
Comment Moderation 169

14. Application Programming Interfaces 175
Introduction to REST 175
Resources Are Everything 176
Request Methods 177
Request and Response Bodies 177
Versioning 178
RESTful Web Services with Flask 179
Creating an API Blueprint 179
Error Handling 180
User Authentication with Flask-HTTPAuth 181
Token-Based Authentication 184
Serializing Resources to and from JSON 186
Implementing Resource Endpoints 188

Pagination of Large Resource Collections 191
Testing Web Services with HTTPie 192

Part III. The Last Mile

15. Testing. 197
Obtaining Code Coverage Reports 197
The Flask Test Client 200
Testing Web Applications 200
Testing Web Services 204
End-to-End Testing with Selenium 205
Is It Worth It? 209

16. Performance. 211
Logging Slow Database Performance 211
Source Code Profiling 213

17. Deployment. 215
Deployment Workflow 215
Logging of Errors During Production 216
Cloud Deployment 217
The Heroku Platform 218
Preparing the Application 218
Testing with Foreman 222
Enabling Secure HTTP with Flask-SSLify 223
Deploying with git push 225
Reviewing Logs 226
Deploying an Upgrade 227
Traditional Hosting 227
Server Setup 227
Importing Environment Variables 228
Setting Up Logging 228

18. Additional Resources. 231
Using an Integrated Development Environment (IDE) 231
Finding Flask Extensions 232
Getting Involved with Flask 232

Index. 235

CHAPTER 1
Installation

Flask is a small framework by most standards, small enough to be called a "micro-framework." It is small enough that once you become familiar with it, you will likely be able to read and understand all of its source code.

But being small does not mean that it does less than other frameworks. Flask was designed as an extensible framework from the ground up; it provides a solid core with the basic services, while *extensions* provide the rest. Because you can pick and choose the extension packages that you want, you end up with a lean stack that has no bloat and does exactly what you need.

Flask has two main dependencies. The routing, debugging, and Web Server Gateway Interface (WSGI) subsystems come from Werkzeug, while template support is provided by Jinja2. Werkzeug and Jinja2 are authored by the core developer of Flask.

There is no native support in Flask for accessing databases, validating web forms, authenticating users, or other high-level tasks. These and many other key services most web applications need are available through extensions that integrate with the core packages. As a developer, you have the power to cherry-pick the extensions that work best for your project or even write your own if you feel inclined to. This is in contrast with a larger framework, where most choices have been made for you and are hard or sometimes impossible to change.

In this chapter, you will learn how to install Flask. The only requirement you need is a computer with Python installed.

The code examples in this book have been verified to work with Python 2.7 and Python 3.3, so using one of these two versions is strongly recommended.

Using Virtual Environments

The most convenient way to install Flask is to use a virtual environment. A virtual environment is a private copy of the Python interpreter onto which you can install packages privately, without affecting the global Python interpreter installed in your system.

Virtual environments are very useful because they prevent package clutter and version conflicts in the system's Python interpreter. Creating a virtual environment for each application ensures that applications have access to only the packages that they use, while the global interpreter remains neat and clean and serves only as a source from which more virtual environments can be created. As an added benefit, virtual environments don't require administrator rights.

Virtual environments are created with the third-party *virtualenv* utility. To check whether you have it installed in your system, type the following command:

```
$ virtualenv --version
```

If you get an error, you will have to install the utility.

Python 3.3 adds native support of virtual environments through the venv module and the pyvenv command. `pyvenv` can be used instead of virtualenv, but note that virtual environments created with pyvenv on Python 3.3 do not include pip, which needs to be installed manually. This limitation has been removed in Python 3.4, where `pyvenv` can be used as a complete virtualenv replacement.

Most Linux distributions provide a package for virtualenv. For example, Ubuntu users can install it with this command:

```
$ sudo apt-get install python-virtualenv
```

If you are using Mac OS X, then you can install virtualenv using easy_install:

```
$ sudo easy_install virtualenv
```

If you are using Microsoft Windows or any operating system that does not provide an official virtualenv package, then you have a slightly more complicated install procedure.

Using your web browser, navigate to *https://bitbucket.org/pypa/setuptools*, the home of the `setuptools` installer. In that page, look for a link to download the installer script. This is a script called *ez_setup.py*. Save this file to a temporary folder on your computer, then run the following commands in that folder:

```
$ python ez_setup.py
$ easy_install virtualenv
```

The previous commands must be issued from an account with administrator rights. On Microsoft Windows, start the command prompt window using the "Run as Administrator" option. On Unix-based systems, the two installation commands must be preceded with `sudo` or executed as the root user. Once installed, the virtualenv utility can be invoked from regular accounts.

Now you need to create the folder that will host the example code, which is available from a GitHub repository. As discussed in "How to Work with the Example Code " on page xiii, the most convenient way to do this is by checking out the code directly from GitHub using a Git client. The following commands download the example code from GitHub and initialize the application folder to version "1a," the initial version of the application:

```
$ git clone https://github.com/miguelgrinberg/flasky.git
$ cd flasky
$ git checkout 1a
```

The next step is to create the Python virtual environment inside the *flasky* folder using the virtualenv command. This command has a single required argument: the name of the virtual environment. A folder with the chosen name will be created in the current directory and all files associated with the virtual environment will be inside. A commonly used naming convention for virtual environments is to call them *venv*:

```
$ virtualenv venv
New python executable in venv/bin/python2.7
Also creating executable in venv/bin/python
Installing setuptools.............done.
Installing pip...............done.
```

Now you have a *venv* folder inside the *flasky* folder with a brand-new virtual environment that contains a private Python interpreter. To start using the virtual environment, you have to "activate" it. If you are using a bash command line (Linux and Mac OS X users), you can activate the virtual environment with this command:

```
$ source venv/bin/activate
```

If you are using Microsoft Windows, the activation command is:

```
$ venv\Scripts\activate
```

When a virtual environment is activated, the location of its Python interpreter is added to the `PATH`, but this change is not permanent; it affects only your current command session. To remind you that you have activated a virtual environment, the activation command modifies the command prompt to include the name of the environment:

```
(venv) $
```

When you are done working with the virtual environment and want to return to the global Python interpreter, type `deactivate` at the command prompt.

Installing Python Packages with pip

Most Python packages are installed with the *pip* utility, which virtualenv automatically adds to all virtual environments upon creation. When a virtual environment is activated, the location of the pip utility is added to the `PATH`.

If you created the virtual environment with pyvenv under Python 3.3, then pip must be installed manually. Installation instructions are available on the pip website. Under Python 3.4, pyvenv installs pip automatically.

To install Flask into the virtual environment, use the following command:

```
(venv) $ pip install flask
```

With this command, Flask and its dependencies are installed in the virtual environment. You can verify that Flask was installed correctly by starting the Python interpreter and trying to import it:

```
(venv) $ python
>>> import flask
>>>
```

If no errors appear, you can congratulate yourself: you are ready for the next chapter, where you will write your first web application.